Quantitative Network Monitoring with NetQRE

Yifei Yuan[1,2], Dong Lin[1,3], Ankit Mishra[1], Sajal Marwaha[1],
Rajeev Alur[1], and Boon Thau Loo[1]

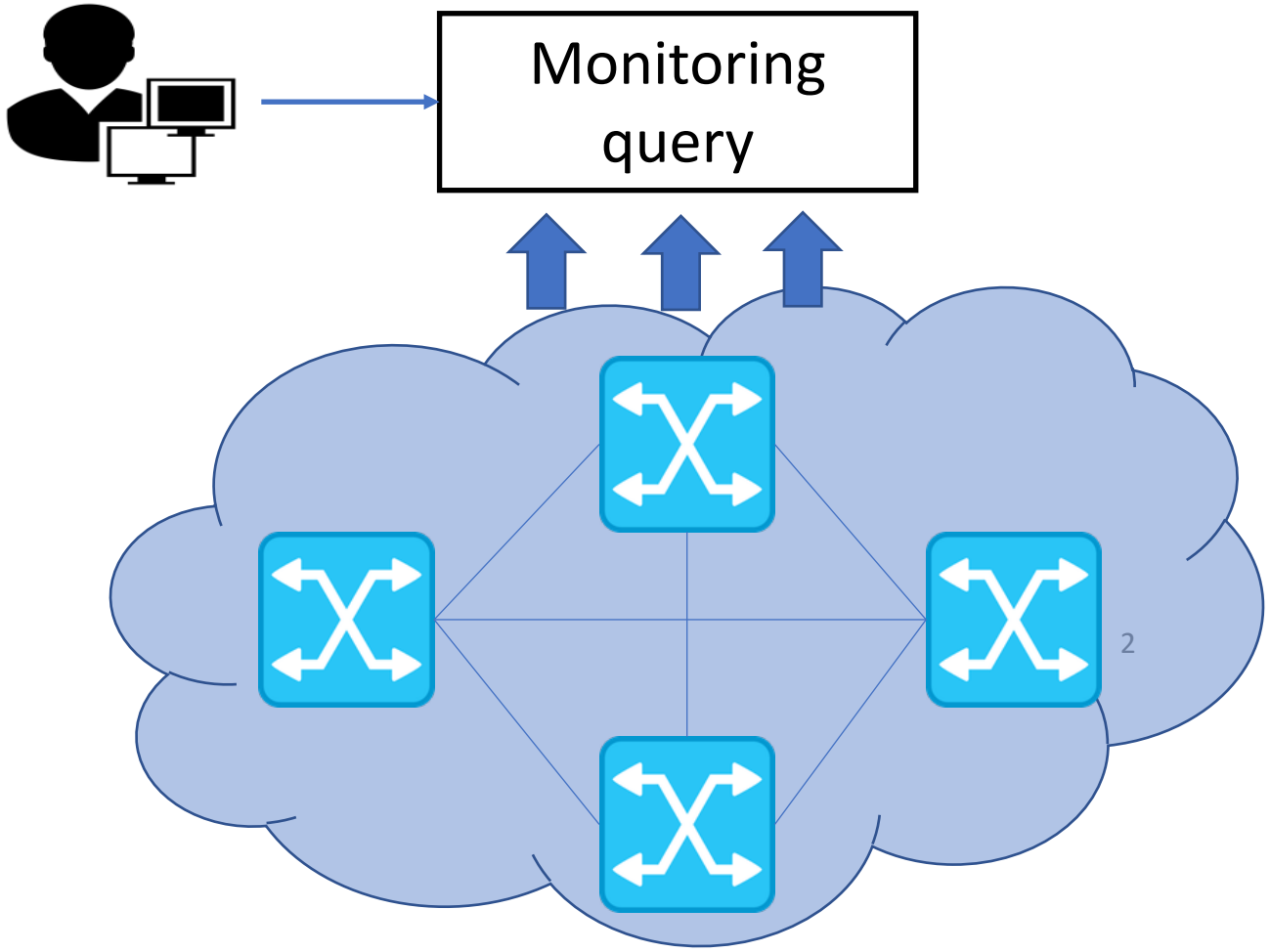
[1]University of Pennsylvania

[2]Carnegie Mellon University

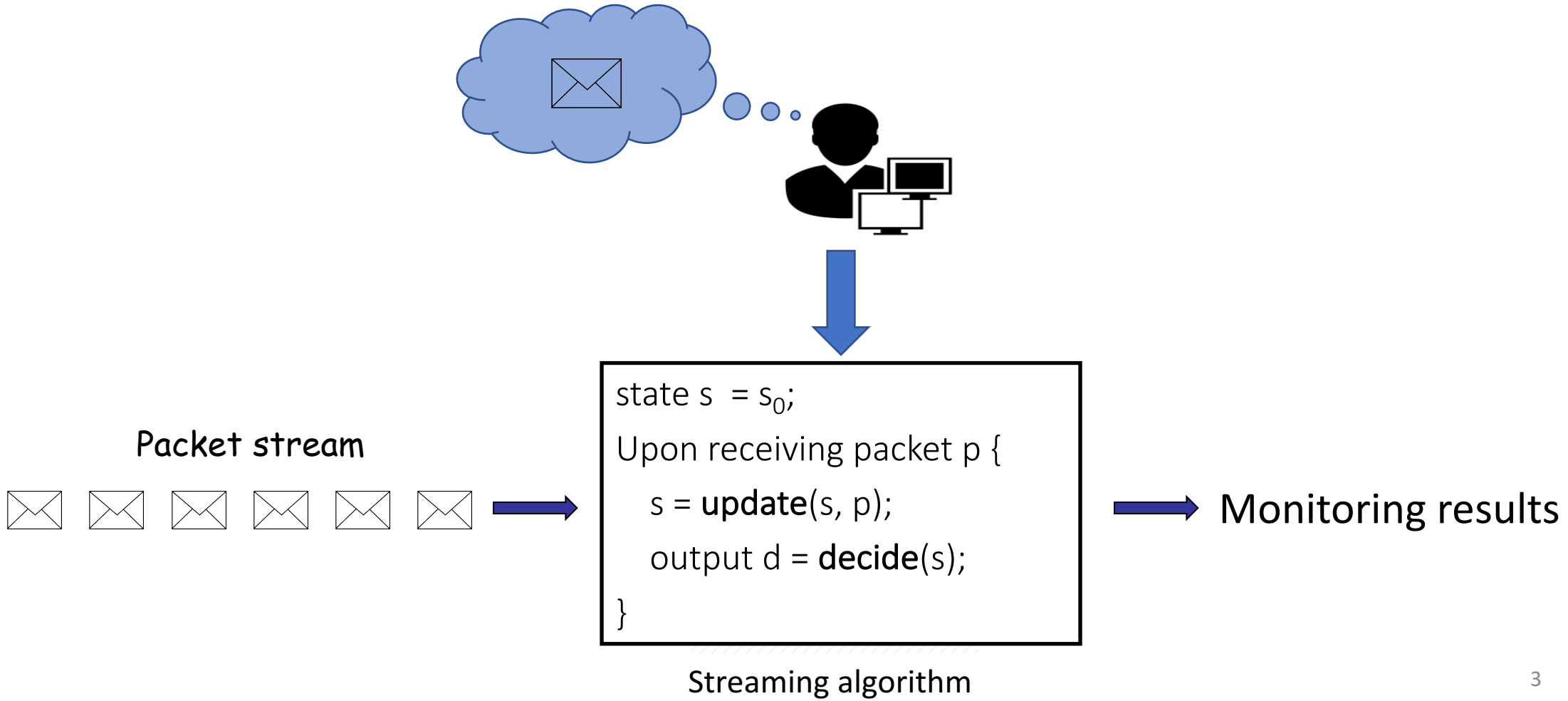
[3]LinkedIn Inc.

Network Monitoring is Important

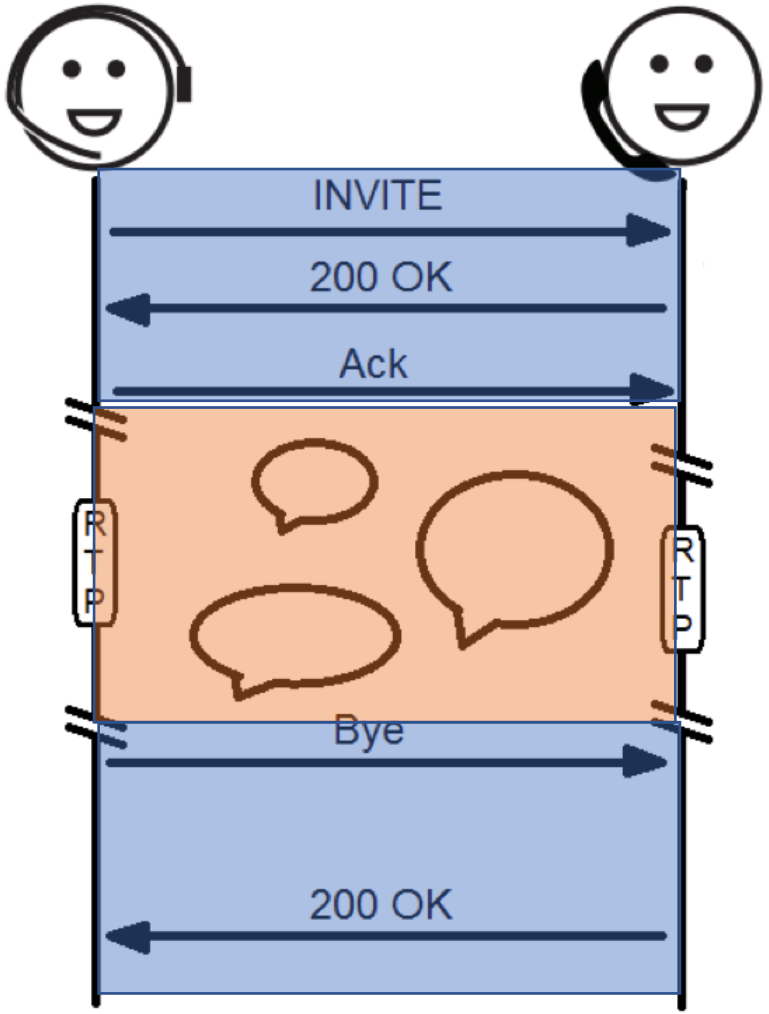
- Security
  - Heavy hitter
  - Super spreader
  - Syn flood
  - Slowloris
  - …
- Performance
  - Traffic matrix
  - Application usage
  - …



Today's Low-level Programming Abstraction



Motivating Example: VoIP Monitoring



Example Policy:

1. Monitor average number of VoIP calls per user
2. Alert a user, if her/his number of calls exceeds a threshold

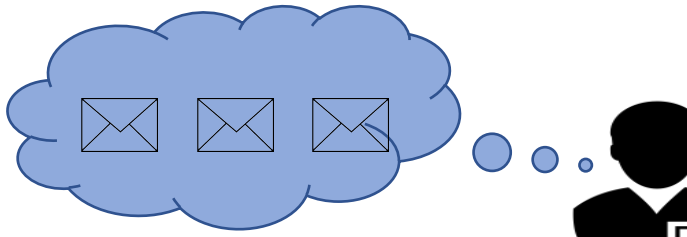
Stateful: Need to maintain state to track VoIP sessions with each incoming packet

Quantitative: Need to compute numerical aggregate based on metrics of past history and across users

?

What low-level state to maintain?
How to update it?

NetQRE Overview



Stream-level abstraction

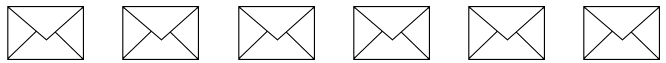
Modular programming

Policy Specification in NetQRE

NetQRE Compiler

- Automatic state inference
- Efficient state update
- Automatic optimization

Packet stream



Monitoring query implementation

Monitoring results

Outline

- Motivation
- NetQRE language
- NetQRE compiler
- Implementation
- Evaluation

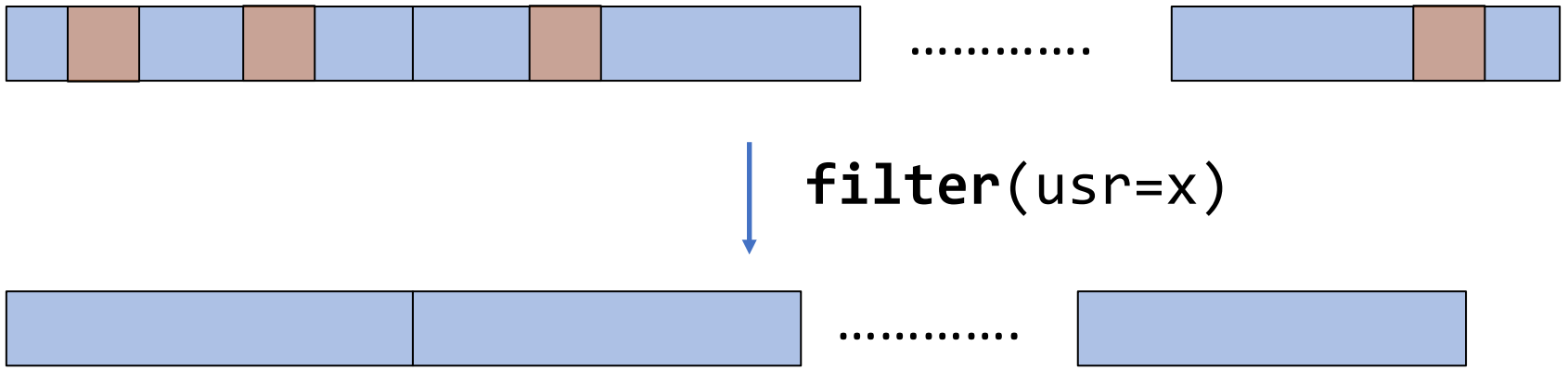
Outline

- Motivation
- NetQRE language
- NetQRE compiler
- Implementation
- Evaluation

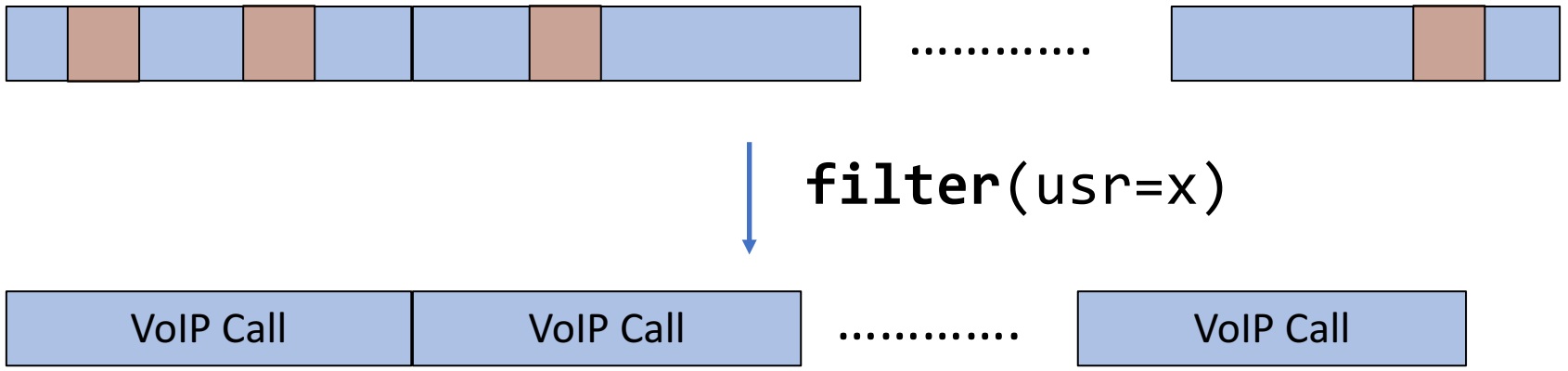
Modular Programming of VoIP Monitoring

- Input: packet stream from all users
- Output: average number of VoIP calls per user
- Procedure:
  - Step 1: Focus on the packet stream from an arbitrary user x
  - Step 2: View the stream as a sequence of calls, and identify each call
  - Step 3: Aggregate across all calls in the stream of the user
  - Step 4: Aggregate across all users

Step 1: Focus on Packet Stream of User x

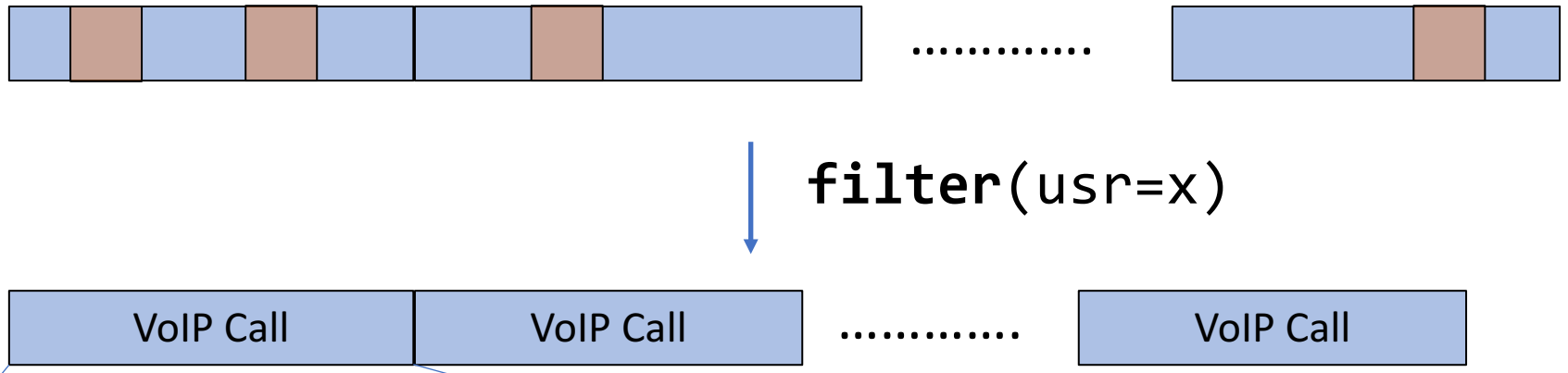


Step 2: Identify A Call



How to specify the pattern of a VoIP call?

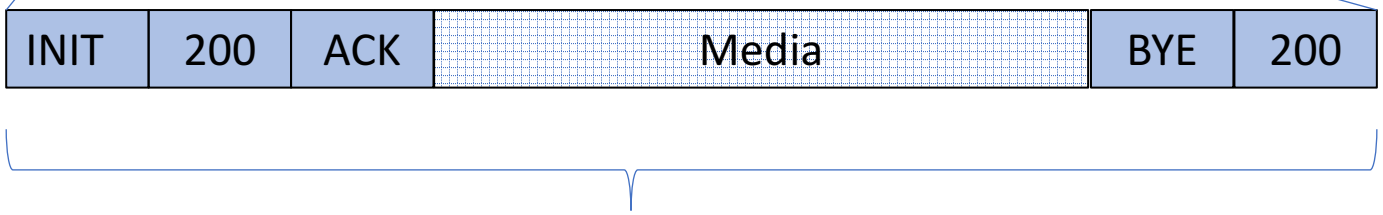
Step 2: Identify A Call



Legend

Packets from user x

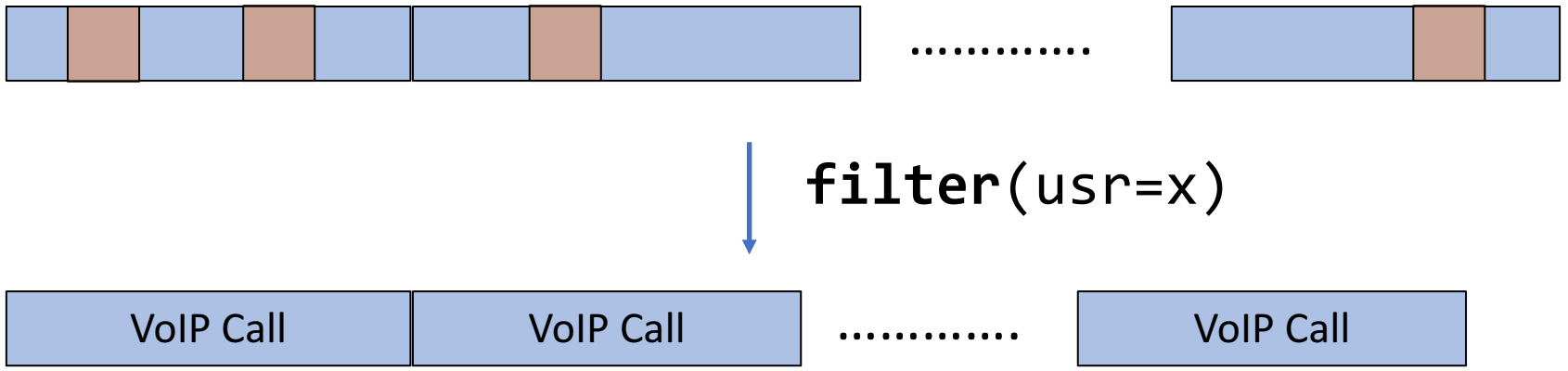
Packets from other users



regular pattern

re_call = [invite] [200] [ack] [data]* [BYE] [200]

Step 2: Identify A Call



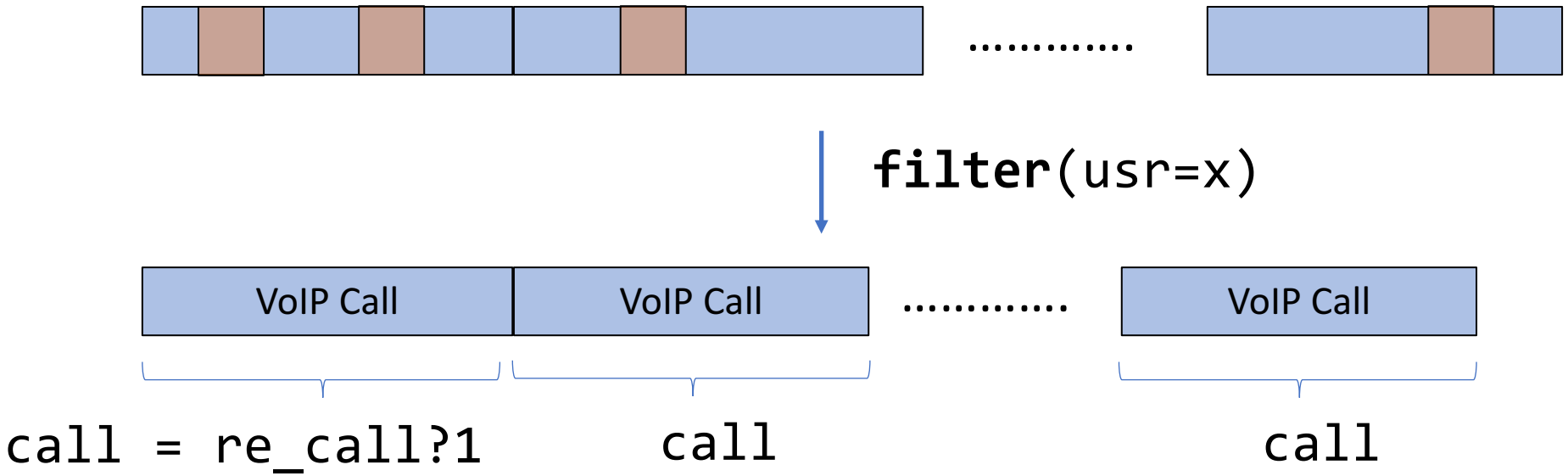
Legend

Packets from user x

Packets from other users

How to associate a numerical value with each pattern?

Step 2: Identify A Call

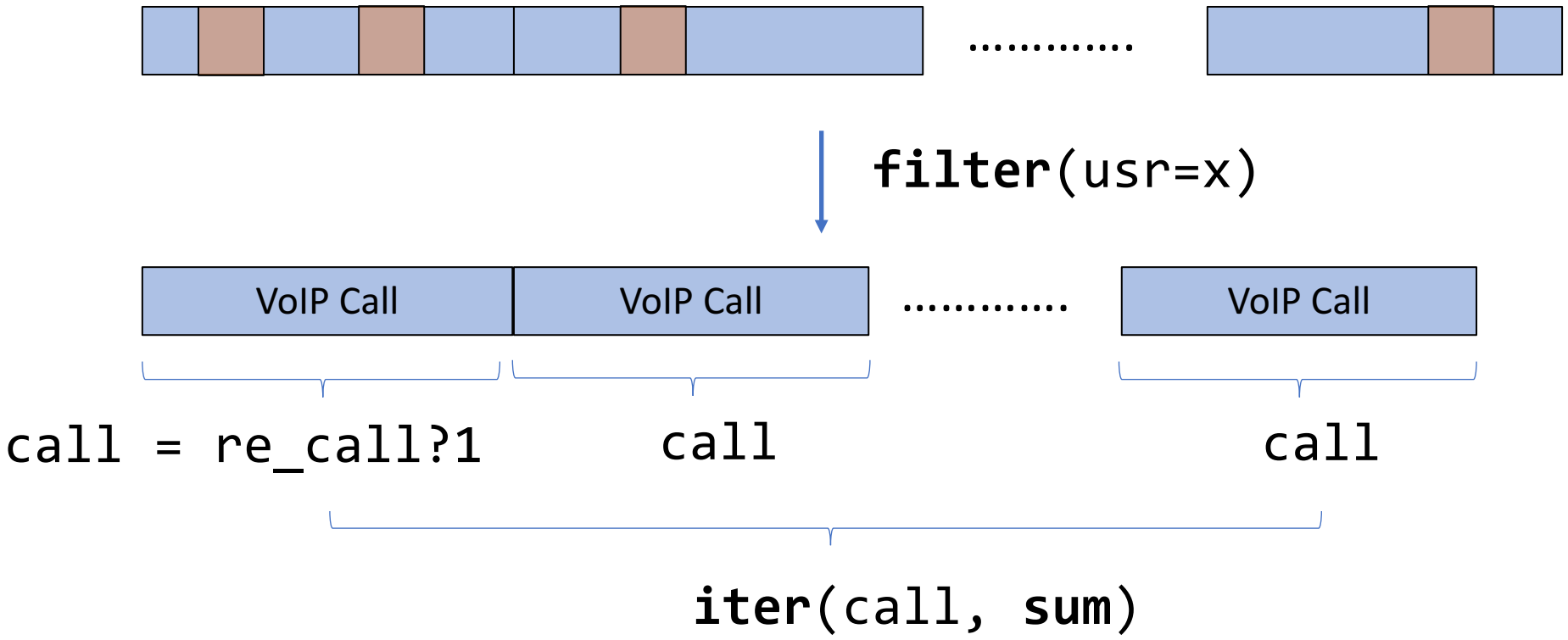


Legend

Packets from user x

Packets from other users

Step 3: Aggregate over All Calls



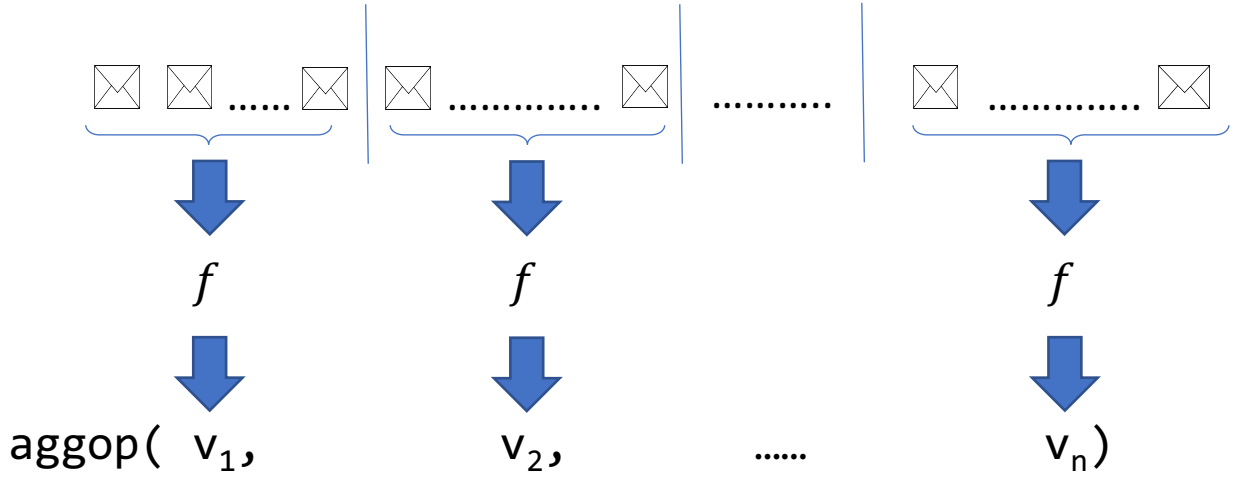
Legend

Packets from user x

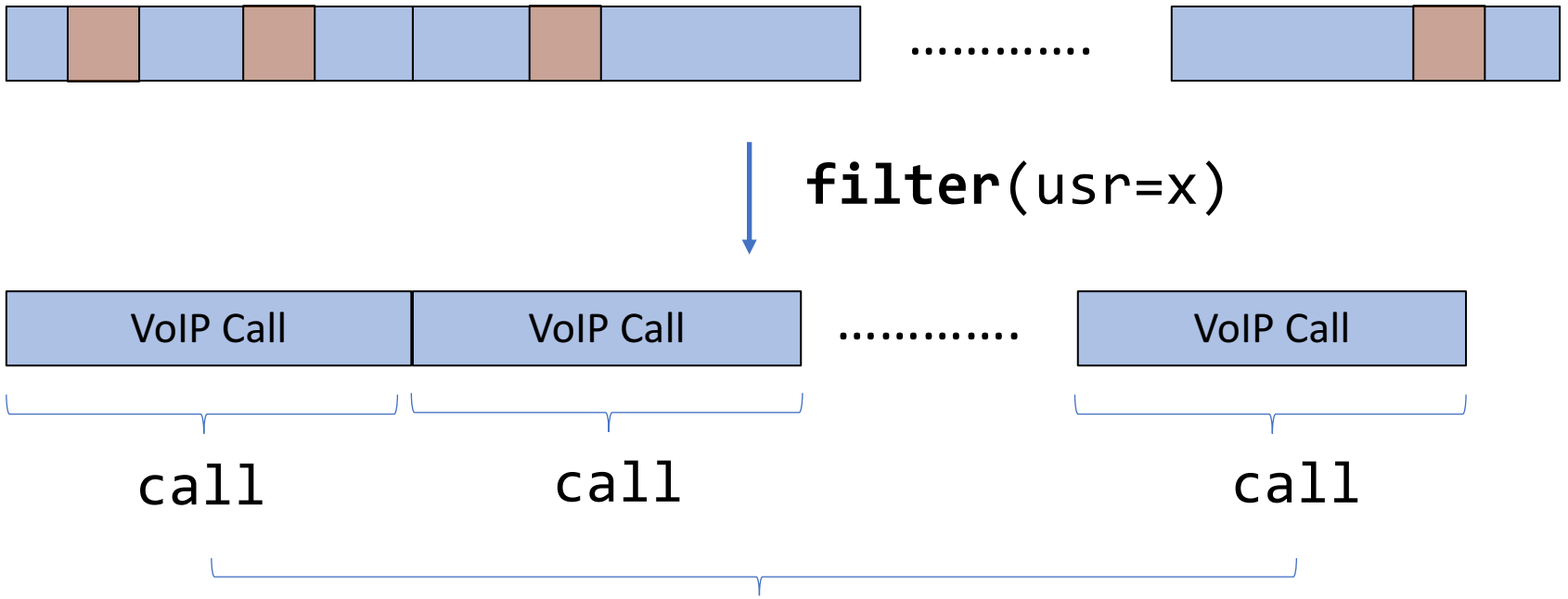
Packets from other users

Stream Iteration: `iter(f, aggop)`

- `f` is a NetQRE function
- `aggop` is an aggregation operator, such as `sum`, `avg`, `max`, `min`.
- Split the stream into multiple substreams $s_1, \ldots, s_n$ such that `f` is defined on each s_i
- Returns $\texttt{aggop}(f(s_1), \ldots, f(s_n))$



Step 4: Aggregation over All Users



Legend

Packets from user x

Packets from other users

```
call_usr(x) = iter(call, sum)

avg{ call_usr(x) | User x }
```

Requirements & Key Ideas

Requirements	Key Ideas
Pattern matching for recognizing traffic patterns	Regular expression (RE) for pattern matching
Handle arbitrary & unknown value	Parametric extension to RE
Quantitative aggregations	Quantitative extension to RE

NetQRE Language

Regular Expression

- Atoms: letters
  - E.g. a, b, …
- Base RE: atoms
- Union: $f \mid g$
- Concatenation: $f \circ g$
- Kleene star: f^*

Parametric extension

Quantitative extension

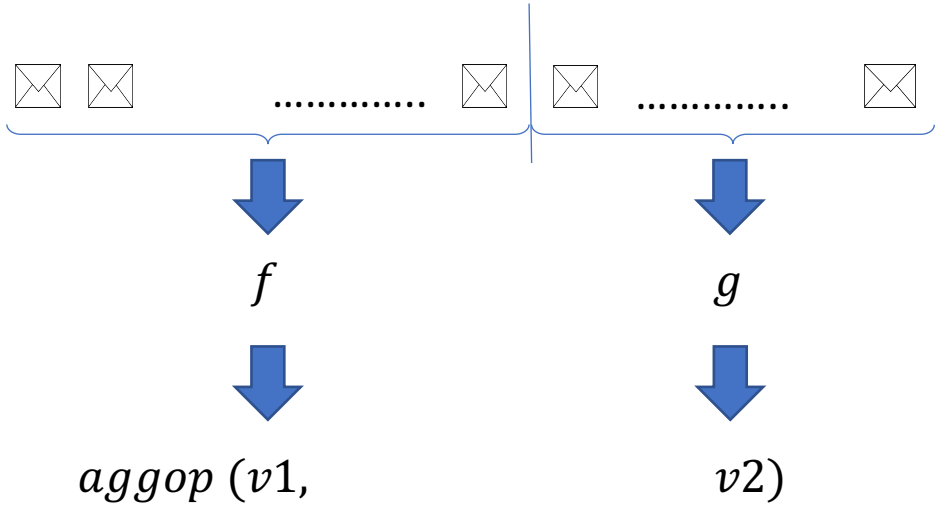
NetQRE

- Atoms: predicate over packets
  - E.g. [srcip==x], [dstip!=10.0.0.1]…
- Base NetQRE: `re ? v`
- Choice: `re ? f : g`
- Split: **`split`**`(f, g, aggop)`
- Iteration: **`iter`**`(f, aggop)`
- Aggregation over parameter:
  `aggop{ f(x) | Type x}`
- Streaming composition: `f >> g`

Details in the paper

Stream Split: **`split(f, g, aggop)`**

- `f` and `g` are two NetQRE functions
- `aggop` is an aggregation operator, such as `sum`, `avg`, `max`, `min`.
- Split the stream into two substreams s_1 and s_2, such that `f` is defined on s_1 and `g` is defined on s_2
- Returns aggop(f(s_1), g(s_2))



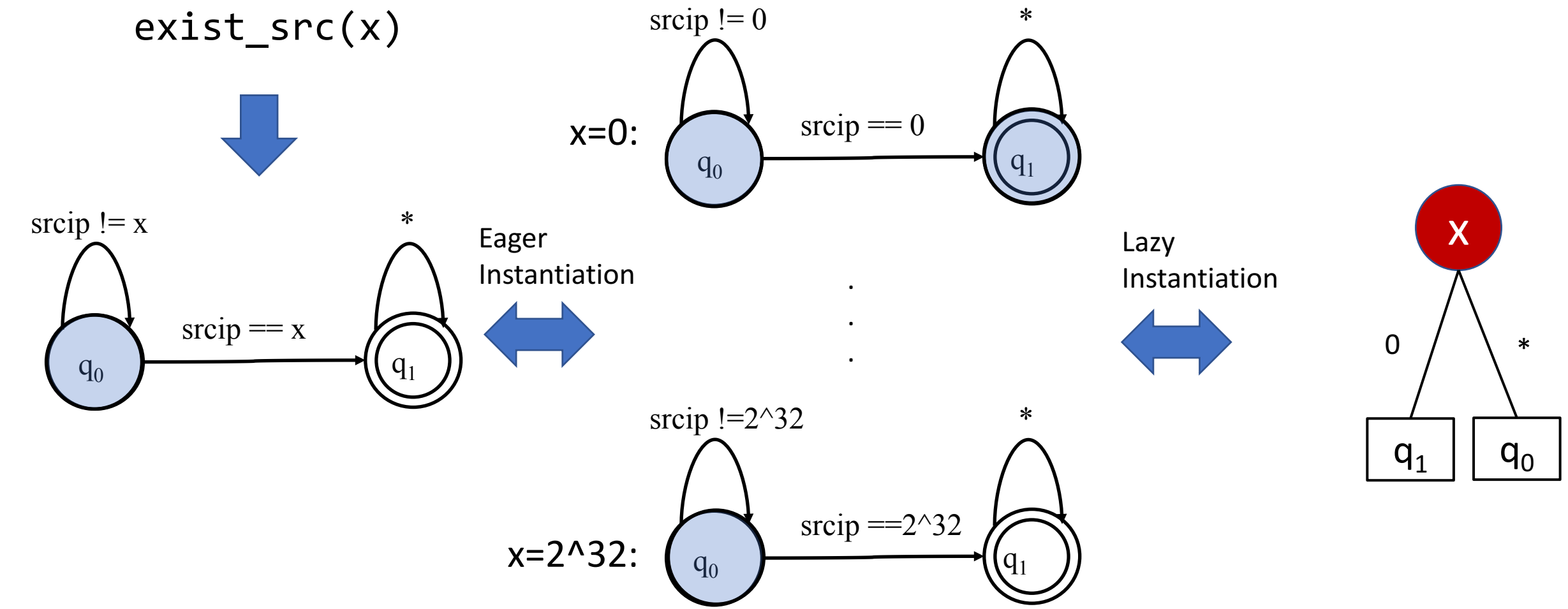
Outline

- Motivation
- NetQRE language
- NetQRE compiler
- Implementation
- Evaluation

NetQRE Compilation

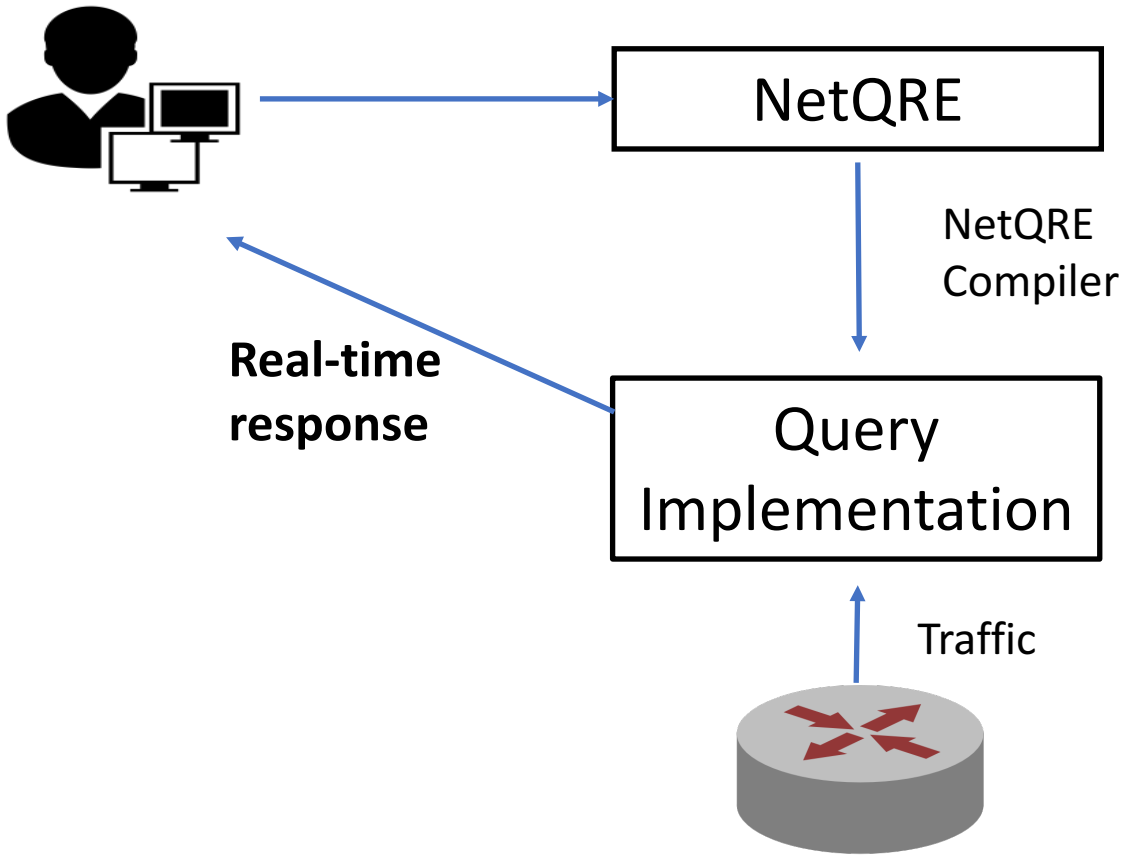
- Goal: Evaluating a query online with small state
  - Independent of length of packet stream
- Insight: Leverage compilation of regular expression to DFA
- Question 1: How to handle parameters?
  - Insight: Lazy instantiation
- Question 2: How to evaluate `split(f,g,aggop)` and `iter(f,aggop)` online?
  - Insight: Keep all possible (but bounded number of) cases
  - Details in the paper

Compilation of RE with Parameters



Implementation

- Single-node deployment
- Compiler implemented in C++
- Compiled code C++
- pcap library for packet capturing



Outline

- Motivation
- NetQRE language
- NetQRE compiler
- Implementation
- Evaluation

Evaluation

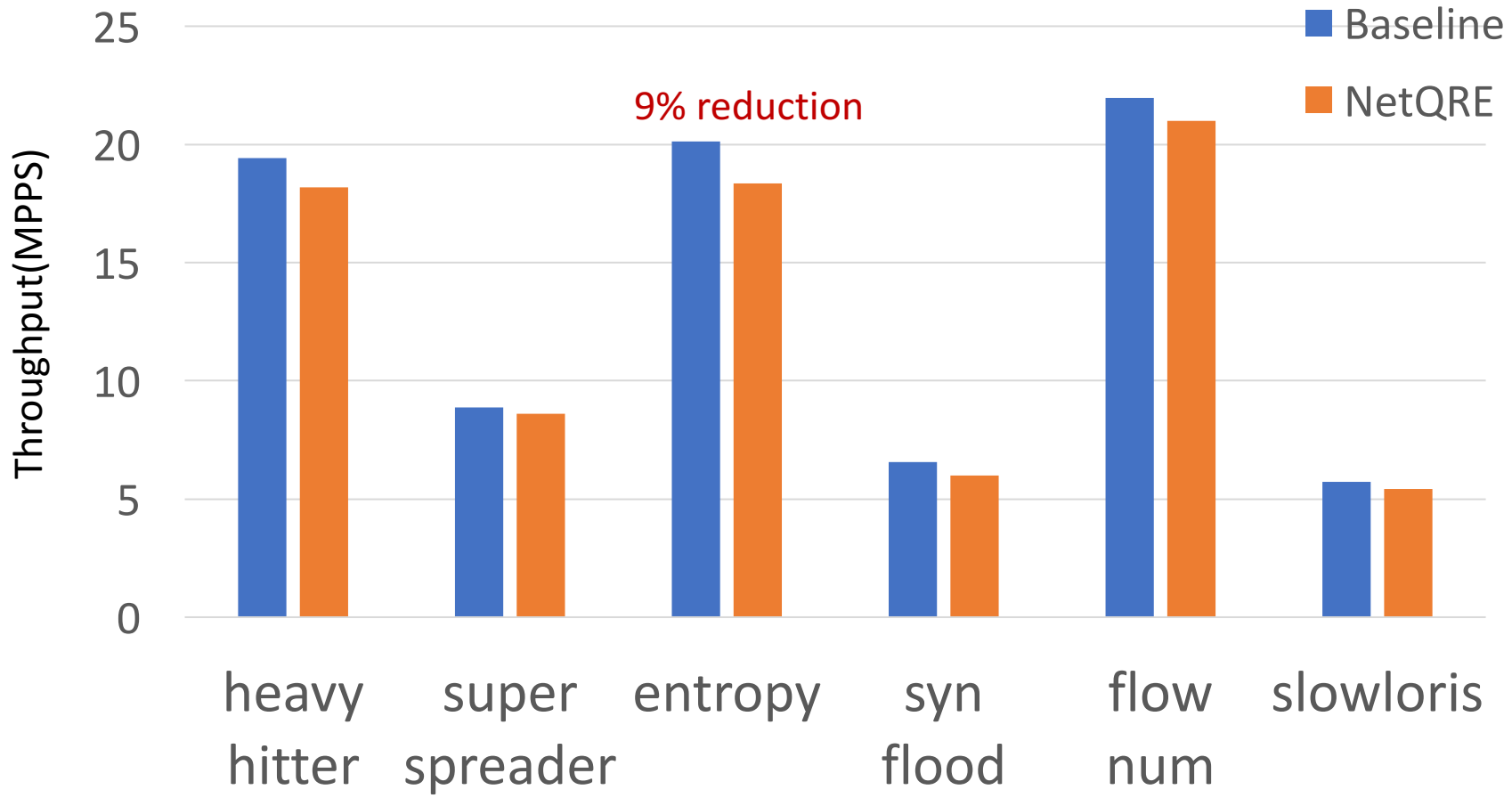
- Is the NetQRE language expressive?
- Is the NetQRE compiled implementation efficient?

Evaluation: Expressiveness

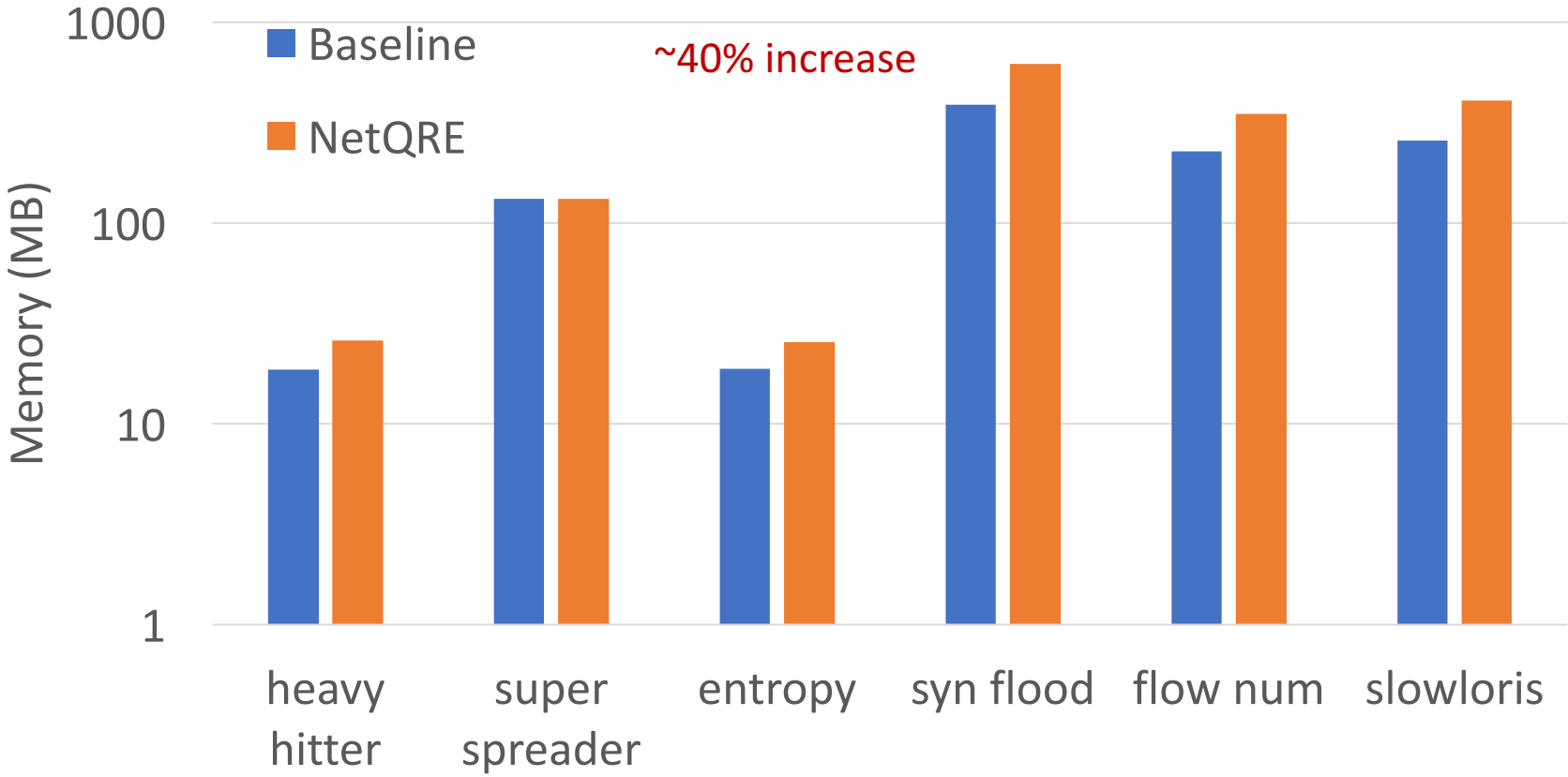
	LoC
Heavy hitter	6
Super spreader	2
Entropy estimation	6
Flow size distribution	8
Traffic change detection	10
Count traffic	2
Completed flows	6
SYN flood detection	9
Slowloris detection	12
Connection lifetime	8
Newly opened connection	11
# duplicated ACKs	5
# VoIP calls	7
VolP usage	18
DNS tunnel detection	4
DNS amplification	4

- Expressive
- Concise
  - 100-1000+ LoC in compiled implementation
  - 100+ LoC in manual implementation

Evaluation: Throughput



Evaluation: Memory



Conclusion

- Motivation: Network monitoring needs high-level abstractions
- Contributions:
  - Stream-level programming abstraction
  - Parametric and quantitative extension to regular expressions
  - Expressive to capture a wide range of monitoring policies
  - Compiled implementation efficient in both throughput and memory
- Future work:
  - Hardware implementation
  - Distributed deployment of NetQRE programs